COMSC-210 Term Project

Instructor: Prof. Robert Burns | rburns@dvc.edu

SOLVING REAL-LIFE PROBLEMS WITH DATA STRUCTURES

The purpose of this project is for COMSC 210 students to demonstrate a working knowledge of data structures for solving real-life problems. Real-life solutions have to be programmed quickly and they have to execute fast. There are three such solutions that you will develop here.

To overcome some of the inefficiencies in code and runtime from past versions of the DvcSchedule series, you'll apply the STL's map template. You'll also answer a question that has been raised about the integrity of the schedule history database, in a separate CPP. And because of a new request from the college administrators, you'll write an app that lets a user type a course name, and your program will output the last time that course was offered at DVC.

Maps are "dictionaries" that store key-value pairs, retrievable knowing only the key. Maps support iteration, and in doing so, they return values *in order* according to the key values.

This last version of DvcSchedule is a remake of lab 5b, which parsed the DVC schedule database and outputted the subject codes and courses offered, listing how many of each since the year 2000. This time, the program is to take *full* advantage of the STL map template, and include nothing that is not required when applying the map. That means no structs and no sorting, because they are not necessary.

Mark Twain is credited with this quote: "If I'd had more time, I would have written a shorter letter". Writing the term project program is the same way -- if you plan carefully and use the full power of the STL, you can solve this in the shortest amount of code. The result should be an elegant solution to the problem, with a lot less code than was used in previous DVC schedule parsers written for this course.

Here are the requirements for the two-part COMSC-210 term project:

REQUIREMENTS (Part 1 of 3)
Write the final version of DvcSchedule.

1. Rewrite DvcSchedule5b.cpp from lab 5, renaming it as **DvcScheduleFinal.cpp**.
2. As in lab 5, parse the dvc-schedule.txt file, located in your working directory.
3. As in lab 5, include NO user input except an optional "press ENTER to continue" at the end if you so wish -- do NOT prompt for the input file name or anything else.
4. Unlike lab 5, do NOT output an Excel xls file.
5. Use only the STL map, set, and pair templates and their iterators -- no other data structures or STL algorithms.
6. Do NOT use programmer-written structs or classes (unlike lab 5).
7. The program should be short enough to fit fully in main -- no functions (except for identifying couts).
8. Code blocks should be labeled with comments, and the programming should be well-organized and professional in its appearance.
9. Including normal code block spacing and commenting, and without multiple statements on single lines, the whole solution should be less than 80 lines of code, not counting your identifying comments and couts, and not counting blank lines.
10. The program should read its input file and execute, all in less than one second (on a classroom PC), before its output begins to appear on the screen.

11. The format of the output should be the same as for the lab 5 version.
12. Remember to skip duplicate section-term entries.

HINTS
The two structs used in the lab 5 version of this solution each consisted of just two data elements. Map declarations also have two data elements. So instead of vectors of struct-based, two-field objects, use maps. As lab 5 had vectors inside a vector, this solution will have maps inside a map.

Maps handle the sorting automatically. The key value in all maps used in this solution will be strings, so the iteration order will be alphanumeric by default.

Look up documentation on the map template, and write test programs to explore its capabilities. Make sure that you understand how its iterators work, and what the elements `first` and `second` are. Make sure that you understand how the member function `find` works -- maps do NOT use the stand-alone "find" algorithm.

The first time you run your program, it may take several seconds to read the input file. But after that, the file should be "cached" in memory, and reading it should be much faster.

Use the online discussion group to discuss possible approaches and solutions with your classmates.

REQUIREMENTS (Part 2 of 3)
Each section-term combination is supposed to be unique -- that is, there should never be two separate entries with the same section and term, but with different course names. It's okay for COMSC-210-8375 in fa2015 to appear more than once -- that's a duplicate and we skip the extra entries. But it's *NOT* ok to have both COMSC-210-8375 *AND* CNT-210-8375 in fa2015. There can be "cross-listed" courses, like ARCHI-126 and ENGIN-126 in Fall 2013 in the same room at the same time with the same instructor (Tumlin, MW 6:00-8:50pm ET-124), but they should have different section numbers, like ARCHI-126-*8349* and ENGIN-126-*8346*.

The DVC Admissions Office suspects that there are at least a few such invalid entries in their database, but their staff is having trouble using Excel to locate them. They heard about what we are studying in COMSC-210, and they want to know if we can help determine if there are any such entries, and what they are.

Write **DvcScheduleCheck.cpp** to solve the problem. Output how many term-section pairs there are with multiple courses associated with them. If there are none, output a message to that effect. But if there are any, output them in a format that you think DVC Admissions could understand.

REQUIREMENTS (Part 3 of 3)
College catalogs sometimes list courses that have not been offered in many years, and departments have no intention of offering them again. It's a disservice to students and to the community to continue to list them. For this (and several other reasons) the administration wants a program that will tell the last semester in which a course was offered.

Write **DvcScheduleSearch.cpp** to solve the problem. In a loop, prompt the user to enter a course name (like COMSC-210) including a subject code (like COMSC), a dash, and an alphanumeric sequence number. If a user enters uppercase or lowercase X, quit the loop and the program -- be sure to explain this in the prompt. For any other input, look up the course and output the last time that course was offered, nicely labeled (like "COMSC-210 was last offered in Fall 2014"). Or if the course is not found, output something like this: "CS=1000 has not been offered since the year 2000" because that's as far back as the database goes.

The response to user input should be practically instantaneous -- so there's no time to open and read the TXT file for every inquiry. Also remember that Spring comes before Summer comes before Fall.

SPECIAL SCORING RULE

Redos are allowed just as in weekly lab assignments. But note that the point deduction for a redo is 5 instead of 2 after the last lab's due date.

WHAT TO SUBMIT

The three CPP program files are to be posted to the COMSC server, in the **project** folder. The posting is due as indicated in the course outline. It is worth 50 points.